file:///C:/A8D3DA8C/SQLSERVER-ProgrammingWithTSqlSql.htm

**Attachment:** Attachment

**Attachment:** Attachment

**Attachment:** Attachment

-----------------------------------------------------------

-- SQL Server 2000 Bible

-- Wiley Publishing

-- Paul Nielsen

-- Chapter  12 - Programming with T-SQL

-----------------------------------------------------------

-----------------------------------------------------------

-- executing a stored procedure with EXEC

sp\_help

EXEC sp\_help

-- the is an ANSI style comment

Select FirstName, LastName   -- selects the columns

  FROM Person                     -- the source table

  Where LastName Like 'Hol\*' -- the row restriction

-- C style comments

/\*

Order table Insert Trigger

Paul Nielsen

ver 1.0 Sept 1, 1998

Logic: etc.

ver 1.1: Nov. 19, 1998

\*/

Set NoCount ON

/\*

go

\*/

-- Debug Commands

Select 3

Print 6

Print 'Begining'

waitfor delay '00:00:02'

Print 'Done'

-------------------------------------------------

-- Variables

-- Declaring variables

DECLARE  @Test INT,

         @TestTwo NVARCHAR(25)

SELECT @Test, @TestTwo

SET @Test = 1

SET @TestTwo = 'a value'

SELECT @Test, @TestTwo

go --terminates the variables

SELECT @Test as BatchTwo, @TestTwo

-- Set and Select

USE Family

SELECT PersonID FROM Person

-- multiple rows & multiple columns

Declare @TempID INT,

           @TempLastName VARCHAR(25)

SET @TempID = 99

SELECT @TempID = PersonID,

    @TempLastName = LastName

  FROM Person

  ORDER BY PersonID

SELECT @TempID, @TempLastName

-- No rows returned

Declare @TempID INT,

           @TempLastName VARCHAR(25)

SET @TempID = 99

SELECT @TempID = PersonID,

    @TempLastName = LastName

  FROM Person

  WHERE PersonID = 100

  ORDER BY PersonID

SELECT @TempID, @TempLastName

-- same as

Declare @TempID INT,

           @TempLastName VARCHAR(25)

SET @TempID = 99

IF 1=0

SELECT @TempID = PersonID,

    @TempLastName = LastName

  FROM Person

  ORDER BY PersonID

SELECT @TempID, @TempLastName

-- Variables within a query

USE OBXKites

DECLARE @ProductCode CHAR(10)

SET @ProductCode = '1001'

SELECT ProductName

  FROM Product

  WHERE Code = @ProductCode

---------------------------------------------------------

-- Procedural Flow

-- If

If 1 = 0

  Print 'Line One'

Print 'Line Two'

-- IF Exists

USE OBXKITES

IF EXISTS(SELECT \* FROM [ORDER] WHERE Closed = 0)

  BEGIN

    Print 'Process Orders'

  END

-- While

Declare @Temp Int

Set @Temp = 0

While @Temp <3

  Begin

    Print 'tested condition' + Str(@Temp)

    Set @Temp = @Temp + 1

  End

-- goto

GOTO ErrorHandler

Print 'more code'

ErrorHandler:

Print 'Logging the error'

-----------------------------------------------------

-- Examining SQL Server with Code

-- sp\_help

sp\_help price

-- Global Variables

Select @@Connections

Select @@CPU\_Busy

Select @@Cursor\_Rows

Select @@DateFirst

Select @@DBTS

Select @@Error

Select @@Fetch\_Status

Select @@Identity

Select @@Idle

Select @@IO\_Busy

Select @@LangID

Select @@Language

Select @@Lock\_TimeOut

Select @@Max\_Connections

Select @@Max\_Precision

Select @@Nestlevel

Select @@Options

Select @@Pack\_Received

Select @@Pack\_Sent

Select @@Packet\_Errors

Select @@ProcID

Select @@RemServer

Select @@RowCount

Select @@ServerName

Select @@ServiceName

Select @@SPID

Select @@TextSize

Select @@TimeTicks

Select @@Total\_Errors

Select @@Total\_Read

Select @@Total\_Write

Select @@TranCount

Select @@Version

---------------------------------------------------

-- Temp Tables & Table Varia

bles

CREATE TABLE #ProductTemp (

  ProductID INT PRIMARY KEY

  )

SELECT Name

  FROM TempDB.dbo.SysObjects

  WHERE Name Like '#Pro%'

-- Global Temp Table

IF NOT EXISTS(

  SELECT \* FROM Tempdb.dbo.Sysobjects

    WHERE Name = '##TempWork')

CREATE TABLE ##TempWork(

  PK INT,

  Col1 INT

)

-- Table Variable

DECLARE @WorkTable TABLE (

  PK INT PRIMARY KEY,

  Col1 INT NOT NULL)

INSERT INTO @WorkTable (PK, Col1)

  VALUES ( 1, 101)

SELECT PK, Col1

  FROM @WorkTable

----------------------------------------------------------------

-- Dynamic SQL

-- Simple EXEC

USE Family

EXEC ('Select LastName from Person Where PersonID = 12')

-- sp\_executeSQL

EXEC sp\_executeSQL

  N'Select LastName from Person Where PersonID = @PersonSelect',

  N'@PersonSelect INT',

  @PersonSelect = 12

-----------------------

-- Developing Dynamic SQL

USE OBXKites

DECLARE

  @SQL NVARCHAR(1024),

  @SQLWhere NVARCHAR(1024),

  @NeedsAnd BIT,

-- User Parameters

  @ProductName VARCHAR(50),

  @ProductCode VARCHAR(10),

  @ProductCategory VARCHAR(50)

-- Initilize Variables

SET @NeedsAnd = 0

SET @SQLWhere = ''

-- Simulate User's Requirements

SET @ProductName = NULL

SET @ProductCode = 1001

SET @ProductCategory = NULL

-- Assembly Dynamic SQL

-- Set up initial SQL Select

IF @ProductCategory IS NULL

  SET @SQL = 'Select ProductName from Product'

ELSE

  SET @SQL = 'Select ProductName from Product

                        Join ProductCategory

                        on Product.ProductCategoryID

                        = ProductCategory.ProductCategoryID'

-- Build the Dynamic Where Clause

IF @ProductName IS NOT NULL

  BEGIN

    SET @SQLWhere = 'ProductName = ' + @ProductName

    SET @NeedsAnd = 1

  END

IF @ProductCode IS NOT NULL

  BEGIN

    IF @NeedsAnd = 1

      SET @SQLWhere = @SQLWhere + ' and '

    SET @SQLWhere = 'Code = ' + @ProductCode

    SET @NeedsAnd = 1

  END

IF @ProductCategory IS NOT NULL

  BEGIN

    IF @NeedsAnd = 1

      SET @SQLWhere = @SQLWhere + ' and '

    SET @SQLWhere = 'ProductCategory = ' + @ProductCategory

    SET @NeedsAnd = 1

  END

-- Assemble the select and the where portions of the dynamic SQL

IF @NeedsAnd = 1

  SET @SQL = @SQL + ' where ' + @SQLWhere

Print @SQL

EXEC sp\_executeSQL @SQL

  WITH RECOMPILE

--------------------------------------------------

-- Recursive Select Variables

-- Denormalizing a List

USE CHA2

DECLARE

  @EventDates VARCHAR(1024)

SET @EventDates = ''

SELECT @EventDates = @EventDates + CONVERT(VARCHAR(15), a.d,107 ) + ';  '

      FROM (Select DateBegin as [d] FROM Event

        JOIN Tour

          ON Event.TourID = Tour.TourID

        WHERE Tour.[Name] = 'Outer Banks Lighthouses') as a

SELECT Left(@EventDates, Len(@EventDates)-1) AS 'Outer Banks Lighthouses Events'

-- Dynamic Crosstab query

USE TempDB

DECLARE   @XColumns NVARCHAR(1024)

SET @XColumns = ''

SELECT @XColumns = @XColumns  + ' SUM(Case X WHEN ''' + [a].[Column] + ''' THEN Data ELSE 0 END) AS ' + [a].[Column] + ','

  FROM

    (SELECT DISTINCT X as [Column]

      FROM RawData  ) as a

SET @XColumns = 'SELECT Y,' + @XColumns + ' SUM(Data) as Total FROM RawData GROUP BY Y ORDER BY Y'

SELECT @XColumns

EXEC sp\_executesql @XColumns

---------------------------------------------------

-- Cursors

-- Cursor default scope

ALTER DATABASE Family SET CURSOR\_DEFAULT LOCAL

SELECT  DATABASEPROPERTYEX('Family', 'IsLocalCursorsDefault')

-- This cursor example is for comparison with the Recursive Select Variable code

-- List the event dates for a tour

--Denormalizing a List with a Cursor

USE CHA2

-- Check the data

SELECT DateBegin

  FROM Event

    JOIN Tour

      ON Event.TourID = Tour.TourID

    WHERE Tour.[Name] = 'Outer Banks Lighthouses'

-- The cursor batch

USE CHA2

DECLARE

  @EventDates VARCHAR(1024),

  @EventDate DATETIME,

  @SemiColon BIT

SET @Semicolon = 0

SET @EventDates = ''

DECLARE cEvent CURSOR FAST\_FORWARD

  FOR SELECT DateBegin

      FROM Event

        JOIN Tour

          ON Event.TourID = Tour.TourID

        WHERE Tour.[Name] = 'Outer Banks Lighthouses'

  OPEN cEvent

  FETCH cEvent INTO @EventDate  -- prime the cursor

  WHILE @@Fetch\_Status = 0

    BEGIN

      IF @Semicolon = 1

        SET @EventDates = @EventDates + '; ' + Convert(VARCHAR(15), @EventDate, 107 )

      ELSE

        BEGIN

          SET @EventDates = Convert(VARCHAR(15), @EventDate,107 )

          SET @SEMICOLON = 1

        END

        FETCH cEvent INTO @EventDate  -- fetch next

    END

  CLOSE cEvent

DEALLOCATE cEvent

SELECT @EventDates

--Building a Crosstab with Dynamic SQL and a Cursor

-- Build the sample data (code from Chapter 10)

USE TempDB

IF EXISTS(SELECT \* FROM SysObjects WHERE Name = 'RawData')

  DROP TABLE RawData

go

CREATE TABLE RawData (

  X VARCHAR(2),

  Y VARCHAR(2),

  Data INT )

go

INSERT RawData (X,Y,Data)

  VALUES( 'A', 'X', 1)

INSERT RawData (X,Y,Data)

  VALUES( 'B', 'X', 2)

INSERT RawData (X,Y,Data)

  VALUES( 'C', 'X', 3)

INSERT RawData (X,Y,Data)

  VALUES( 'A', 'Y', 4)

INSERT RawData (X,Y,Data)

  VALUES( 'B', 'Y', 5)

INSERT RawData (X,Y,Data)

  VALUES( 'D', 'Y', 6)

INSERT RawData (X,Y,Data)

  VALUES( 'A', 'Z', 7)

INSERT RawData (X,Y,Data)

  VALUES( 'B', 'Z', 8)

INSERT RawData (X,Y,Data)

  VALUES( 'C', 'Z', 9)

INSERT RawData (X,Y,Data)

  VALUES( 'D', 'Z', 10)

INSERT RawData (X,Y,Data)

  VALUES( 'A', 'X', 2)

INSERT RawData (X,Y,Data)

  VALUES( 'A', 'X', 3)

INSERT RawData (X,Y,Data)

  VALUES( 'A', 'Y', 2)

INSERT RawData (X,Y,Data)

  VALUES( 'A', 'Y', 5)

INSERT RawData (X,Y,Data)

  VALUES( 'A', 'Y', 1)

INSERT RawData (X,Y,Data)

  VALUES( 'D', 'Y', 50)

-- check the data

SELECT \* FROM RawData

-- Fixed Column CrossTab with Y Subtotal

USE TempDB

SELECT Y,

  SUM(Case X WHEN 'A' THEN Data ELSE 0 END) AS A,

  SUM(Case X WHEN 'B' THEN Data ELSE 0 END) AS B,

  SUM(Case X WHEN 'C' THEN Data ELSE 0 END) AS C,

  SUM(Case X WHEN 'D' THEN Data ELSE 0 END) AS D,

  SUM(Data) as Total

  FROM RawData

  GROUP BY Y

  ORDER BY Y

-- CrossTab with Dynamic X Using Cursor

DECLARE

  @XColumns NVARCHAR(1024),

  @XColumn VARCHAR(50),

  @SemiColon BIT

SET @Semicolon = 0

SET @XColumns = ''

DECLARE ColNames CURSOR FAST\_FORWARD

  FOR

  SELECT DISTINCT X as [Column]

    FROM RawData

    ORDER BY X

  OPEN ColNames

  FETCH ColNames INTO @XColumn

  WHILE @@Fetch\_Status = 0

    BEGIN

        SET @XColumns = @XColumns  + ', SUM(Case X WHEN ''' + @XColumn + ''' THEN Data ELSE 0 END) AS ' + @XColumn

        FETCH ColNames INTO @XColumn  -- fetch next

    END

  CLOSE ColNames

DEALLOCATE ColNames

SET @XColumns = 'SELECT Y' + @XColumns + ', SUM(Data) as Total FROM RawData GROUP BY Y ORDER BY Y'

EXEC sp\_executesql  @XColumns

-- Navigating a Tree with a Recursive Cursor

USE Family

--Check the data

SELECT

    Person.FirstName + ' ' + IsNull(Person.SrJr,'') as Grandfather,

    Gen1.FirstName  + ' ' +  IsNull(Gen1.SrJr,'') as Gen1,

    Gen2.FirstName  + ' ' +  IsNull(Gen2.SrJr,'') as Gen2

  FROM Person

    Left JOIN Person Gen1

      ON Person.PersonID = Gen1.FatherID

    left JOIN Person Gen2

      ON Gen1.PersonID = Gen2.FatherID

  WHERE Person.PersonID = 2

  ORDER BY Person.DateofBirth, Gen1.DateOfBirth, Gen2.DateofBirth

go

-- Set the cursor scope to local to prevent the recursive cursor from bombing

ALTER DATABASE Family SET CURSOR\_DEFAULT LOCAL

SELECT  DATABASEPROPERTYEX('Family', 'IsLocalCursorsDefault')

go

-- the Recursive Cursor

-- For the current person examine all the children

CREATE PROCEDURE ExamineChild (@

ParentID INT)

AS

SET Nocount On

DECLARE @ChildID INT,

  @Childname VARCHAR(25)

DECLARE cChild CURSOR LOCAL FAST\_FORWARD

  FOR SELECT PersonID, Firstname + ' ' + LastName + ' ' + IsNull(SrJr,'') as PersonName

      FROM Person

      WHERE Person.FatherID = @ParentID

        OR Person.MotherID = @ParentID

      ORDER BY Person.DateOfBirth

  OPEN cChild

  FETCH cChild INTO @ChildID, @ChildName  -- prime the cursor

  WHILE @@Fetch\_Status = 0

    BEGIN

      PRINT

        SPACE(@@NestLevel \* 2) + '+ '

          + Cast(@ChildID as VARCHAR(4)) + ' '

          + @ChildName

      -- Recursively find the grandchildren

      EXEC ExamineChild @ChildID

      FETCH cChild INTO @ChildID, @ChildName

    END

  CLOSE cChild

DEALLOCATE cChild

-- there will be an error while creating this procedure

-- because SQL Server can not establish a dependency

-- between ExamineChild on ExamineChild

-- because ExamineChild does not yet exist when the

-- procedure is created.

-- execute the recursive cursor stored procedure

EXEC ExamineChild 2

--------------------------------------

-- Navigating a Recursive Tree using a Set-based solution

CREATE TABLE #FamilyTree (

  PersonID INT,

  Generation INT,

  FamilyLine VarChar(25) Default ''

  )

delete #FamilyTree

DECLARE

  @Generation INT,

  @FirstPerson INT

SET @Generation = 1

SET @FirstPerson = 2

-- prime the temp table with the top person(s) in the queue

INSERT #FamilyTree (PersonID, Generation, FamilyLine)

  SELECT @FirstPerson, @Generation, @FirstPerson

WHILE @@RowCount > 0

  BEGIN

    SET @Generation = @Generation + 1

    INSERT #FamilyTree (PersonID, Generation, FamilyLine)

      SELECT Person.PersonID,

             @Generation,

             #FamilyTree.FamilyLine

             + ' ' + Str(Person.PersonID,5)

        FROM Person

          JOIN #FamilyTree

            ON #FamilyTree.Generation = @Generation - 1

              AND

              (Person.MotherID = #FamilyTree.PersonID

                OR

               Person.FatherID = #FamilyTree.PersonID)

  END

SELECT PersonID, Generation, FamilyLine

  FROM #FamilyTree

  Order by FamilyLine

SELECT SPACE(Generation \* 2) + '+ '

          + Cast(#FamilyTree.PersonID as VARCHAR(4)) + ' '

          + FirstName + ' ' + LastName + ' ' + IsNull(SrJr,'')

          AS FamilyTree

  FROM #FamilyTree

    JOIN Person

      ON #FamilyTree.PersonID = Person.PersonID

  ORDER BY FamilyLine

----------------------------------------------------------

-- Error Handling

USE Family

UPDATE Person

  SET PersonID = 1

  Where PersonID = 2

Print @@Error

Print @@Error

-- saving @@error to alocal variable

USE Family

DECLARE @err INT

UPDATE Person

  SET PersonID = 1

  Where PersonID = 2

SET @err = @@Error

IF @err <> 0

  Begin

    -- error handling code

    Print @err

  End

-- Using @RowCount

USE FAMILY

UPDATE Person

  SET LastName = 'Johnson'

  WHERE PersonID = 100

IF @@RowCount = 0

  Begin

    -- error handling code

    Print 'no rows affected'

  End

-- Fatal Errors

SELECT Error, Severity, Description

  FROM Master.dbo.SysMessages

  WHERE Severity > 16

-- Rasierror simple form

RAISERROR 5551212  'Unable to update Customer'

-- Raiserror Windows Form

RAISERROR('Unable to update Customer', 14, 1)

-- Severity

RAISERROR('Print', 10,1)

RAISERROR('Info', 14,1)

RAISERROR('Warning', 15,1)

RAISERROR('Critical', 16,1)

--  Messages Arguments

RAISERROR ('Unable to update %s.', 14, 1, 'Customer')

-- Stored Messages

EXEC sp\_addmessage 50001, 16, 'Unable to update %s'

EXEC sp\_addmessage 50001, 16, 'Still unable to update %s', @Replace = 'Replace'

SELECT 'EXEC sp\_addmessage, '

    + Cast(Error as VARCHAR(7))

    + ', ' + Cast(Severity as VARCHAR(2))

    + ', ''' + [Description] +  ''''

  FROM Master.dbo.SysMessages

  WHERE Error > 50000

EXEC sp\_dropmessage 50001

-- With Log

RAISERROR ('Unable to update %s.', 14, 1, 'Customer')

  WITH LOG